Average-Case Completeness in Tag Systems

Cook, Matthew ; Neary, Turlough

DOI: https://doi.org/10.4230/LIPIcs.STACS.2019.20

Average-Case Completeness in Tag Systems

Matthew Cook
University of Zürich, Switzerland
ETH Zürich, Switzerland
cook@ini.ethz.ch

Turlough Neary
University of Zürich, Switzerland
ETH Zürich, Switzerland
tneary@ini.ethz.ch

Abstract

To prove average-case NP-completeness for a problem, we must choose a known average-case complete problem and reduce it to that problem. Unfortunately, the set of options to choose from is far smaller than for standard (worst-case) NP-completeness. In an effort to help remedy this we focus on tag systems, which due to their extreme simplicity have been a target for other types of reductions for many problems including the matrix mortality problem, the Post correspondence problem, the universality of cellular automaton Rule 110, and all of the smallest universal single-tape Turing machines. Here we show that a tag system can efficiently simulate a Turing machine even when the input is provided in an extremely simple encoding which adds just $\log n$ carefully set bits to encode an arbitrary Turing machine input of length n. As a result we show that the bounded halting problem for nondeterministic tag systems is average-case NP-complete. This result is unexpected when one considers that in the current state of the art for simple universal systems it had appeared that there was a trade-off whereby simpler systems required more complicated input encodings. In other words, although simple systems can compute interesting things, they had appeared to require very carefully encoded inputs in order to do so. Our result surprisingly goes in the opposite direction by giving the first average-case completeness result for such a simple model of computation. In ongoing work we have already found applications of our result having used it to give average-case NP-completeness results for a 2D generalization of the Collatz function, a nondeterministic version of the 2D elementary functions studied by Koiran and Moore, 3D piecewise affine maps, and bounded Post correspondence problem instances that use simpler word pairs than previous results.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness

Keywords and phrases average-case NP-completeness, encoding complexity, tag system, bounded halting problem

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.20

Funding *Turlough Neary*: Supported by Swiss National Science Foundation grant numbers 200021-153295 and 200021-166231.

1 Introduction

Given the massive interest in worst-case NP-completeness, the literature devoted to proving the stronger result of average-case NP-completeness can be considered quite limited. This is surprising when one considers the practical importance of determining whether or not we are likely to encounter intractable instances in problems we wish to solve. One reason for the smaller number of results is that it is more difficult to prove the stricter form of reduction required to show average-case NP-completeness [11, 25]. A first step towards overcoming this difficulty is to give new average-case completeness results for problems whose simplicity allows for easier average-case reductions to other systems.

Of all the simple models in the literature where a new average-case completeness result would have applications to a wide range of problems, perhaps the most compelling case can be made for tag systems, a very simple form of rewriting system introduced by Post [18]. The simplicity of the context free rewrite rule employed by tag systems has made them a favored target of simulation by many other systems. For this reason, tag systems have been used either through individual reductions or via chains of reductions to prove many undecidability and hardness results (e.g. [4, 9, 12, 21, 22, 23, 24, 26]). Reductions to tag systems have also yielded significant improvements in lower bounds for a number of well studied problems [14, 16, 19, 20]. So proving that the bounded halting problem for tag systems is average-case NP-complete could lead to other new average-case NP-completeness results and even improved lower bounds for existing results. In fact we [5] have already used 2-tag systems to give an average-case NP-completeness result for Post correspondence problem instances that use shorter word pairs than those found in [8, 25]. In ongoing work [6] we have already begun using 2-tag systems as the starting point for chains of simulations that prove average-case NP-completeness. We have used 2-tag systems to prove the average-case NP-completeness of a bounded reachability problem for a generalized 2D[1] version of the Collatz function that is nondeterministic [6]. As a corollary of our result we find that a nondeterministic version of the 2D elementary functions of Koiran and Moore [10] also have a bounded reachability problem that is average-case NP-complete. In addition we simulate tag systems to prove an average-case NP-completeness result for bounded reachability in 3D piecewise affine maps that are nondeterministic [6].

It is worth noting that the applications of tag systems given in the references above are not where the applications end; they propagate to other results through further chains of reductions. The results in [16] are an example where binary tag systems were used to significantly improve the undecidability bounds for both the Post correspondence problem and the matrix mortality problem. The new bounds for the matrix mortality problem also give improved undecidability bounds for the problem in [2] of reaching the origin with piecewise linear systems and for the quantum measurements problem in [7]. The results in the present paper are a first step towards proving average case completeness results for bounded versions of these problems.

There are some obvious reasons to think an attempt to prove an average-case completeness result for a system as simple as a tag systems is doomed to fail. It seems natural to expect that the simplest systems require unwieldy encodings to compute, or suffer from an exponential trade-off when it comes to time efficiency, and for a long time the literature seemed to bear this out. However, in [17, 26] Neary and Woods showed that many of the simplest known models of computation [4, 12, 14, 19, 20, 22] actually simulate Turing machines in polynomial time, an exponential improvement over the previous simulations. It follows that many simple systems now have a P-complete prediction problem, which means that there is no known way to predict the long term behavior of these systems significantly faster than by explicit step by step simulation.

Despite these improvements in efficiency it remained the case that the simplest universal systems utilized complicated input encodings [4, 12, 13, 14, 16, 19, 20]. So it seemed reasonable to expect that as programs get shorter or the form of rules get simpler, extra complexity gets forced into the input encoding. This observation was expressed nicely by Yedidia and Aaronson in [27]:

[1] It is an open problem as to whether or not generalized 1D Collatz functions can simulate Turing machines in polynomial time [10] and so proving an NP-completeness result for a nondeterministic generalization of 1D Collatz functions would most likely require some radically new encoding technique.

> "the known small universal Turing machines achieve their small size only at the cost of an extremely complicated description format for the input machine. That is, most of the complexity gets "shunted" from the Turing machine itself to the input encoding format."

The complexity of the input encodings used by the simplest known universal systems means that their input encodings have a very low chance of occurring when sampled from a uniform distribution over the input alphabet. So while many of the simplest systems are now known to have P-complete prediction problems based on carefully encoded inputs, it could nonetheless be the case that the behavior of the simplest universal systems is easy to predict *on average*.

Here we give a first result indicating that this is not the case. Specifically, we show that tag systems can efficiently simulate the computation of binary Turing machines when provided with an extremely simple encoding which adds just $\log n$ carefully set bits to encode an arbitrary input of length n. As a result we find that the bounded halting problem for nondeterministic tag systems is average-case NP-complete.

2 Preliminaries

The length of a word w is denoted by $|w|$. We let ϵ denote the empty word. Given a natural number i we let $\langle i \rangle$ be its binary digit representation.

2.1 2-Tag Systems

▶ **Definition 1.** *A 2-tag system consists of a finite alphabet of symbols Σ and a finite set of rules $R : \Sigma \to \Sigma^*$.*

The computation of a 2-tag system acts on a word $w = \sigma_0\sigma_1 \dots \sigma_l$ which we call the *dataword*. The entire configuration is given by w. In a computation step, the two symbols $\sigma_0\sigma_1$ are deleted and we apply a rule for the first symbol σ_0, i.e., a rule of the form $\sigma_0 \to \sigma_{l+1} \dots \sigma_{l+c}$, by appending the word $\sigma_{l+1} \dots \sigma_{l+c}$. A dataword (configuration) w_2 is obtained from w_1 via a single computation step as follows:

$$\sigma_0\sigma_1\sigma_2 \dots \sigma_l \vdash \sigma_2 \dots \sigma_l\sigma_{l+1} \dots \sigma_{l+c}$$

where $\sigma_0 \to \sigma_{l+1} \dots \sigma_{l+c} \in R$. A 2-tag system *halts* if $|w| < 2$ or if there is no rule defined for the leftmost symbol σ_0 . A *round* is the $\lfloor \frac{|w|}{2} \rfloor$ or $\lceil \frac{|w|}{2} \rceil$ computation steps that traverse the word w exactly once. We say a symbol σ_i of w is *read* if and only if at the start of some computation step it is the leftmost symbol (i.e. i is even, so the rule $\sigma_i \to \sigma_{k+1} \dots \sigma_{k+c}$ will be applied). In this work we consider tag systems that are nondeterministic, that is they are permitted to have more than one rule for each $\sigma_i \in \Sigma$.

In [26] it was shown that 2-tag systems efficiently simulate deterministic binary Turing machines in time $O(t^4(\log t)^2)$ where t is the running time of the Turing machine. This time overhead was later improved in Chapter 5 of [15] to give Theorem 2.

▶ **Theorem 2** (Woods and Neary [26, 15])**.** *Given a single tape deterministic Turing machine M that computes in time t then there is a 2-tag system T_M that simulates the computation of M and computes in time $O(t^2 \log t)$.*

In [15] given a binary input word $w = x_1x_2 \dots x_n$ for Turing machine M it is encoded as the T_M input dataword

$$\bar{x}_1\dot{\bar{x}}_1\, x_2\dot{x}_2\, x_3\dot{x}_3 \dots\, x_n\dot{x}_n\, (aa)^{2^{\lceil \log_2 n \rceil + c}} \tag{1}$$

While the results in [26, 15] offer a polynomial time simulation of Turing machines using this input encoding, this does not allow us to prove an average-case NP-completeness result, as the probability of choosing a word that encodes some w (via Equation (1)) is exponentially smaller than the probability of choosing w. In Lemma 11 we will show how 2-tag systems can compute the encoding in Equation (1) when provided with a more compact encoding of M's input word. Our compact binary encoding requires only $n + \log n$ encoding bits to encode M's input word (the remaining $n + \log n$ bits in our length $2(n + \log n)$ encoding are arbitrary padding bits). This gives an input encoding that is only polynomially less likely to occur than the input to M and so using Lemma 11 and the results in [26] we can prove that the bounded halting problem for tag systems is average-case NP-complete.

2.2 Average-Case Complexity

When we speak of the average-case complexity of a decision problem we are considering the expected time to solve that problem with respect to some distribution over instances of the problem. This leads to the notion of a distributional problem [1, 8, 11, 25]. A distributional problem is a pair (D, μ) where D is a decision problem and μ is a distribution over instances of D. In this work instances of D are given as binary words and we let L_D be the set of instances for which the answer to the problem D is positive.

The definitions in this section are adapted from [1, 8, 11, 25]. For the distributional problems in this work we take the uniform probability distribution [8, 11] for binary words $w \in \{0,1\}^*$ which is proportional to $\mu(w) = |w|^{-2}2^{-|w|}$. The probability of choosing a word of length n is proportional to $1/n^2$. We denote the distribution over all words of length n with μ_n and so we have $\mu_n(w) = 2^{-n}$. We let $Pf(w_1, w_2, \ldots, w_m) \in \{0,1\}^*$ be the prefix free code for the binary words $w_1, w_2, \ldots, w_m$ where $|Pf(u_1, \ldots, u_m)| = 2m + \sum_{i=1}^{m} |w_i| + 2\log_2 |w_i|$ (see [3]).

▶ **Definition 3** (Average polynomial function). *A function f that maps words to natural numbers is polynomial on average with respect to a distribution μ if there exists an $\epsilon > 0$ such that for all n*

$$\sum_{|w|=n} \mu_n(w)(f(w))^\epsilon = O(n)$$

where $w \in \{0,1\}^n$.

▶ **Definition 4** (Average polynomial time). *An algorithm runs in average polynomial time with respect to a distribution μ if its running time is bounded by an average polynomial function with respect to μ.*

Average-case reductions insist that when reducing a distributional problem (D, μ) to another distributional problem (D', μ'), instances $x \in D$ should be at most polynomially more likely than the instances they reduce to. This property is enforced by condition 2 of Definition 5. Previous 2-tag system simulations of Turing machine used input encodings that do not satisfy this condition. In Lemma 11 and Corollary 12 we show that 2-tag system with a concise input encoding can simulate Turing machines efficiently, and then in Theorem 13 we show that this new input encoding satisfies condition 2 of Definition 5.

▶ **Definition 5** (Ptime reduction between distributional problems). *A distributional problem (D, μ)* Ptime reduces *to a distributional problem (D', μ') if there exists a polynomial time computable function $g(w) = y$, where $w \in D$ and $y \in D'$, and a polynomial $p(|w|)$ such that the following two conditions hold:*

1. $g(w) \in L_{D'}$ *if and only if* $w \in L_D$
2. $\sum_{g(w)=y} \mu(w) \leqslant p(|w|)\,\mu'(y)$

▶ **Definition 6** (Dilation of a distributional problem). *A dilation* Δ *of a distributional problem* (D, μ) *is a distributional problem* (D_Δ, μ_Δ) *where instances of* D_Δ *include extra padding. The dilation* Δ *maps each instance* w *of* D *to a set given by* $\{Pf(w, s_w)|s_w \in S_w\}$ *where* S_w *is a finite set of binary words. The set* D_Δ *is given by the union of all sets produced by applying* Δ *to the instances of* D *(i.e.* $D_\Delta = \bigcup_{w \in D} \{Pf(w, s_w)|s_w \in S_w\}$*). For each instance* $Pf(w, s_w)$ *of* D_Δ*,* $Pf(w, s_w) \in L_{D_\Delta}$ *if and only if* $w \in L_D$*. The probability distribution is given by* $\mu_\Delta(Pf(w, s_w)) = \dfrac{\mu(w)2^{-|s_w|}}{\sum_{r' \in S_w} 2^{-|r'|}}$.

We say a dilation is a *Ptime dilation* if it is computed by a randomized algorithm $\mathcal{A}$ that runs in polynomial time, that is on input w $\mathcal{A}$ outputs an element from $\{Pf(w, s_w)|s_w \in S_w\}$ in time polynomial in $|w|$. We think of s_w as the sequence of coin flips made by $\mathcal{A}$.

▶ **Definition 7** (Nonrare dilation). *A dilation* Δ *(as defined in Definition 3) is nonrare if* R_Δ *(given in Equation* (2)*) is polynomial on average with respect to the distribution* μ'*.*

$$R_\Delta(w) = \frac{1}{\sum_{s \in S_w} 2^{-|s|}} \tag{2}$$

▶ **Definition 8** (Ptime randomized reduction between distributional problems). *A distributional problem* (D, μ) *randomly reduces to a distributional problem* (D', μ') *if* (D, μ) *has a nonrare Ptime dilation* (D_Δ, μ_Δ) *such that* (D_Δ, μ_Δ) *Ptime reduces to* (D', μ') *(see Definition 5).*

Let $M_1, M_2, M_3, \ldots$ be an enumeration of nondeterministic binary Turing machines and let $T_1, T_2, T_3, \ldots$ be an enumeration of nondeterministic 2-tag systems.

The problem in Definition 9 is known to be average-case NP-complete and it will be used in our reduction at the end of the next section to prove that the problem for tag systems given in Definition 10 is average-case NP-complete.

▶ **Definition 9** (Distributional bounded halting problem for nondeterministic Turing machines).
Problem: Given a nondeterministic Turing machine M_i*, a binary word* w*, and a natural number* t*, determine whether or not* M *halts in* t *steps when given* w *as input.*
Instance: A binary word $Pf(\langle i \rangle, w, 1^t)$
Distribution: Proportional to $\mu(\langle i \rangle, w, t) = 2^{-(|\langle i \rangle|+|w|)}|\langle i \rangle|^{-2}|w|^{-2}t^{-2}$.

▶ **Definition 10** (Distributional bounded halting problem for nondeterministic 2-tag systems).
Problem: Given a nondeterministic 2-tag system T_i*, a binary word* w*, and a natural number* t*, determine whether or not* T *halts in* t *steps when given* w *as input.*
Instance: A binary word $Pf(\langle i \rangle, w, 1^t)$
Distribution: Proportional to $\mu(\langle i \rangle, w, t) = 2^{-(|\langle i \rangle|+|w|)}|\langle i \rangle|^{-2}|w|^{-2}t^{-2}$.

3 New Concise Input Encoding for 2-Tag Systems

In Equation (3), we define an encoding function $f : \Sigma^n \times \Sigma^{n+\lceil \log_2 n \rceil + c} \to \{0,1\}^*$ where $c = 1$ if $2^{\lceil \log_2 n \rceil} < n + \lceil \log_2 n \rceil$ otherwise $c = 0$. The encoding function takes $w = x_1 x_2 \ldots x_n$ an arbitrary binary Turing machine input and $s = z_1 z_2 \ldots z_{n+\lceil \log_2 n \rceil + c}$ an arbitrary padding

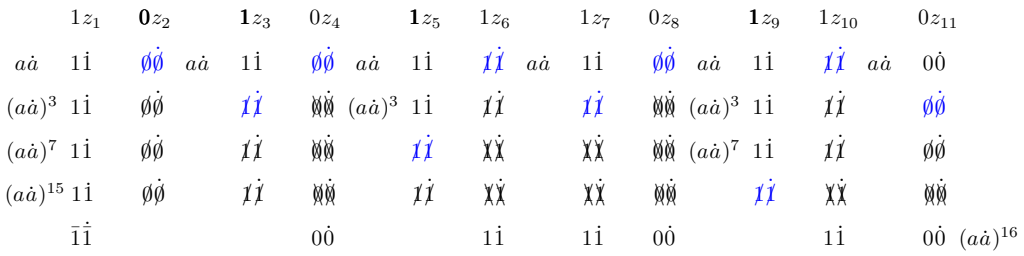

Figure 1 Six datawords giving an overview of the tag system algorithm in Lemma 11. The dataword at the top is the input to the tag system and is the encoding of the word 1011010 via Function (3). The extra parity bits inserted are highlighted in the top row in bold. The extra white space between pairs of symbols and the vertical alignment of symbols are for readability. The $z_i \in \{0,1\}$ are arbitrary padding symbols that are not read by the tag system so the first round eliminates them. The second line from top gives the dataword after one iteration (three rounds) of our algorithm on the input dataword, the third line from the top gives the dataword after two iterations of our algorithm on the input dataword and so on until the fifth dataword where one further round produces the output of the algorithm on the last line. Each iteration of our algorithm involves three rounds of the dataword. A detailed view of the 2 other rounds not shown here is given in Datawords (7) and (8).

word (where $x_i, z_j \in \{0,1\}$) and maps the pair to a 2-tag system input word. The value of $f(w,s)$ is obtained by taking $x_1x_2\ldots x_n$ and inserting either $\lceil \log_2 n \rceil$ or $\lceil \log_2 n \rceil + 1$ extra *parity bits* and then adding a padding bit (z_j) after each bit in the resulting word. An example of the application of f appears in Figure 1.

$$f(w,s) = \underset{1}{u}_1 z_1 \, \underset{1}{u}_2 z_2 \, \ldots \, \underset{1}{u}_{n+\lceil \log_2 n \rceil + c} \, z_{n+\lceil \log_2 n \rceil + c} \begin{cases} z_i \in \{0,1\} & \text{(unread bits)} \\ \underset{1}{u}_i = x_{i-\lceil \log_2 i \rceil} & \text{if } \forall_k \;\; i \neq 2^k + 1 \\ \underset{1}{u}_{2^k+1} = h(w,k) & \text{(parity bits)} \end{cases} \tag{3}$$

where $h(w,k)$ is given by Equation (4), $w_{2^k+1} = \underset{1}{u}_{3\cdot 2^k+1} \underset{1}{u}_{5\cdot 2^k+1} \underset{1}{u}_{7\cdot 2^k+1} \ldots \underset{1}{u}_{m\cdot 2^k+1}$ is a binary word of certain input bits with $m = 2y+1$, $y, k \in \mathbb{N}$, $0 \leqslant k < \lceil \log_2 n \rceil + c$, and $m(2^k) + 1 \leqslant n + \lceil \log_2 n \rceil + c < (m+2)2^k + 1$. Note that each w_{2^k+1} word (there is one for each k) is formed from a set of bits, and these sets are disjoint from each other and from the set of parity bits, but the union of these sets and the set of parity bits is the entire input word (apart from the throw-away z_i bits). The function h indicates how word w_{2^k+1} is used to set parity bit $\underset{1}{u}_{2^k+1}$.

$$h(w,k) = \begin{cases} 0 & \text{if } |w_{2^k+1}| > 0 \text{ and number of 1 symbols in } w_{2^k+1} \text{ is even} \\ 1 & \text{if } |w_{2^k+1}| > 0 \text{ and number of 1 symbols in } w_{2^k+1} \text{ is odd} \\ 1 & \text{if } |w_{2^k+1}| = 0 \end{cases} \tag{4}$$

▶ **Lemma 11.** *Let $w = x_1x_2\ldots x_n$ and $s = z_1z_2\ldots z_{n+\lceil \log_2 n \rceil + c}$ be binary words where $x_i, z_j \in \{0,1\}$. There is a 2-tag system $\mathcal{T}$ that takes a binary word of the form*

$$f(w,s) = \underset{1}{u}_1 z_1 \, \underset{1}{u}_2 z_2 \, \ldots \, \underset{1}{u}_{n+\lceil \log_2 n \rceil + c} \, z_{n+\lceil \log_2 n \rceil + c} \tag{5}$$

as input and produces a word of the form

$$\bar{x}_4 1 \dot{\bar{x}}_4 1 \; x_4 2 \dot{x}_4 2 \; \cdots \; x_4 n \dot{x}_4 n \; (a_4 \dot{a})^{2^{\lceil \log_2 n \rceil + c}} \tag{6}$$

in time $O(n \log_2 n)$.

Proof. We prove the existence of such a tag system $\mathcal{T}$ by exhibiting one and showing why it works, which will take the next few pages.

To produce Dataword (5) from Dataword (6) there are three tasks to be carried out: (a) append $(a_2 \dot{a})^{2^{\lceil \log_2 n \rceil + c}}$ to the right end of the word, (b) change the first pair of symbols be uniquely in the form $\bar{x}_4 \dot{\bar{x}}_4$ and (c) delete the parity symbols (i.e. $u_{2^k+1} \dot{u}_{2^k+1}$ pairs in Equation (3)). We give an overview of how the algorithm achieves these tasks concurrently and then we give the rules for the tag system and explain how they implement this algorithm.

Tasks (a) and (b)

To append a subword of the form $(a\dot{a})^{2^{\lceil \log_2 n \rceil + c}}$ one could append a single $a\dot{a}$ pair at the end of the dataword and then iterate a process where on each iteration each $a\dot{a}$ pair is mapped to $a\dot{a}\, a\dot{a}$ so that after $\lceil \log_2 n \rceil + c$ iterations the initial $a\dot{a}$ pair has grown to become $(a\dot{a})^{2^{\lceil \log_2 n \rceil + c}}$. Unfortunately there is no unique pair in the initial dataword that can be used to append a single $a\dot{a}$ pair and so we must append $a\dot{a}$ pairs throughout the dataword which gives $a\dot{a}$ subwords that grow in multiple locations throughout the dataword. Half of these growing $a\dot{a}$ subwords are deleted on each of the $\lceil \log_2 n \rceil + c$ iterations so that after the last iteration only one subword of the form $(a\dot{a})^{2^{\lceil \log_2 n \rceil + c}}$ remains.

We now explain how our algorithm knows when it has carried out the $\lceil \log_2 n \rceil + c$ iterations needed to grow the $(a\dot{a})^{2^{\lceil \log_2 n \rceil + c}}$ word.

Each iteration involves 3 rounds of the dataword and marks every second unmarked $u\dot{u}$ pair by mapping it to $\not{u}\dot{\not{u}}$. See for example Figure 1 where on the first iteration pairs 2, 4, 6, 8 and 10 are marked, on the second iteration pairs 3, 7 and 11 are marked, and so on. Using the $\lceil \log_2 n \rceil + c$ parity pairs we can determine when we have completed $\lceil \log_2 n \rceil + c$ iterations of this marking process. On iteration k we mark the pairs $u_{2^k+1}\dot{u}_{2^k+1}, u_{3(2^k)+1}\dot{u}_{3(2^k)+1}, u_{5(2^k)+1}\dot{u}_{5(2^k)+1} \ldots u_{m(2^k)+1}\dot{u}_{m(2^k)+1}$ and so on each iteration exactly one parity pair $(u_{2^k+1}\dot{u}_{2^k+1})$ is marked. To see that this is the case it is sufficient to note that if at the start of iteration i the unmarked pairs are

$$u_1\dot{u}_1, \quad u_{2^i+1}\dot{u}_{2^i+1}, \quad u_{2(2^i)+1}\dot{u}_{2(2^i)+1}, \quad u_{3(2^i)+1}\dot{u}_{3(2^i)+1}, \quad \cdots \quad u_{s_1(2^i)+1}\dot{u}_{s_1(2^i)+1}$$

then at the start of iteration $i+1$ the unmarked pairs are

$$u_1\dot{u}_1, \; u_{2^{i+1}+1}\dot{u}_{2^{i+1}+1}, \; u_{2(2^{i+1})+1}\dot{u}_{2(2^{i+1})+1}, \; u_{3(2^{i+1})+1}\dot{u}_{3(2^{i+1})+1}, \; \ldots \; u_{s_2(2^{k+1})+1}\dot{u}_{s_2(2^{k+1})+1}$$

and so pairs of the form $u_{2^k+1}\dot{u}_{2^k+1}, u_{3(2^k)+1}\dot{u}_{3(2^k)+1}, u_{5(2^k)+1}\dot{u}_{5(2^k)+1} \ldots u_{m(2^k)+1}\dot{u}_{m(2^k)+1}$ are marked on iteration k for $k = i$ and $k = i+1$. The value of each $u_{2^k+1}\dot{u}_{2^k+1}$ pair is set via Equation (4) so that an even number of $1\dot{1}$ pairs are marked during each of the

first $\lceil \log_2 n \rceil + c - 1$ iterations, and on iteration number $\lceil \log_2 n \rceil + c$ an odd number of $1\dot{1}$ pairs are marked. So by checking whether the number of $u\dot{u} = 1\dot{1}$ pairs marked during each iteration is odd or even our algorithm can determine when exactly $\lceil \log_2 n \rceil + c$ iterations have been completed.

Now we can explain how we append $(a\dot{a})^{2^{\lceil \log_2 n \rceil + c}}$ during the $\lceil \log_2 n \rceil + c$ iterations described above. On the first iteration a single $a\dot{a}$ pair is appended to the left of each $u\dot{u}$ pair that remains unmarked by applying a rule of the form $u \to a\dot{a}\ u\dot{u}$. On each subsequent iteration each $a\dot{a}$ pair is replaced with two $a\dot{a}$ pairs if the $u\dot{u}$ pair immediately to the right remains unmarked and each $a\dot{a}$ pair is deleted by mapping it to the empty word if the $u\dot{u}$ pair immediately to the right is marked on that iteration. In addition on each iteration each unmarked $u\dot{u}$ pair adds another $a\dot{a}$ pair. So immediately to the left of each unmarked $u\dot{u}$ pair we have a single $a\dot{a}$ pair after the first iteration, 3 $a\dot{a}$ pairs after the second iteration, 7 $a\dot{a}$ pairs after the third iteration, and $2^k - 1$ $a\dot{a}$ pairs after the k^{th} iteration (see Figure 1). Thus after $\lceil \log_2 n \rceil + c$ iterations we have $(a\dot{a})^{2^{\lceil \log_2 n \rceil + c} - 1}$ to the left of the only unmarked pair $(u_1 u_1)$. Then in one final round each $a\dot{a}$ in $(a\dot{a})^{2^{\lceil \log_2 n \rceil + c} - 1}$ is mapped to $a\dot{a}$, and this lone remaining $u_1 u_1$ pair appends one further $a\dot{a}$ to give $(a\dot{a})^{2^{\lceil \log_2 n \rceil + c}}$ at the right end of the dataword, while also producing the $\bar{u}\dot{\bar{u}}$ to simultaneously achieve Tasks (a) and (b).

Task (c)

From Equation (3) the parity pairs that appear in the dataword have the form $u_{2^k+1}\dot{u}_{2^k+1}$ and before we can delete these pairs we must first distinguish them from all other $u_i\dot{u}_i$ pairs in the dataword. Recall from Task (a) that on iteration k we mark the pairs $u_{2^k+1}\dot{u}_{2^k+1}$, $u_{3(2^k)+1}\dot{u}_{3(2^k)+1}$, $u_{5(2^k)+1}\dot{u}_{5(2^k)+1}$, $\ldots$ $u_{m(2^k)+1}\dot{u}_{m(2^k)+1}$. Note that $u_{2^k+1}\dot{u}_{2^k+1}$ is the left-most pair marked during iteration k, and since every second unmarked $u\dot{u}$ pair is marked during iteration k there must be a single unmarked pair to the left of $u_{2^k+1}\dot{u}_{2^k+1}$, and this unmarked pair must be $u_1\dot{u}_1$ since $u_1\dot{u}_1$ is never marked which also means that in all iterations following iteration k there is exactly one unmarked pair to the left of $\not{u}_{2^k+1}\dot{\not{u}}_{2^k+1}$. It is also the case that immediately following iteration k there must be at least $l \geqslant 2$ unmarked pairs to left of each pair of the form $\not{u}_{j(2^k)+1}\dot{\not{u}}_{j(2^k)+1}$ where $j \geqslant 3$. It follows that after a further r iterations (iteration $k + r$) when we have continued marking every second unmarked pair there will be $\frac{\lceil l \rceil}{2^r}$ unmarked pairs to the left of $\not{u}_{j(2^k)+1}\dot{\not{u}}_{j(2^k)+1}$. Since $l \geqslant 2$ and our algorithm iterates until there is only one unmarked pair there is an iteration $k + r$ where $\frac{\lceil l \rceil}{2^r} = 1$ and $\frac{\lceil l \rceil}{2^{r-1}} = 2$. It follows that for all pairs of the form $\not{u}_{j(2^k)+1}\dot{\not{u}}_{j(2^k)+1}$ where $j \geqslant 3$ there is at least one iteration where the number of unmarked pairs to the left of $\not{u}_{j(2^k)+1}\dot{\not{u}}_{j(2^k)+1}$ is even at the beginning of the iteration. If at the beginning of an iteration the number of unmarked pairs to the left of a $\not{u}\dot{\not{u}}$ pair is even then we apply the rule $\not{u} \to \not{\not{u}}\dot{\not{\not{u}}}$ (see for example Figure 1). It follows that pairs of the form $\not{u}_{j(2^k)+1}\dot{\not{u}}_{j(2^k)+1}$ where $j \geqslant 3$ will be changed

Table 1 Tag system rules where $u \in \{0, 1\}$. The left column specifies when the rules are used during the algorithm: during round 1, 2 or 3 of each iteration, or during the final round on the dataword.

round 1	$\underset{1}{u} \to \underset{2\,2\,2}{u\dot{u}\ddot{u}}$, $\underset{1}{a} \to \underset{2\,2}{a\dot{a}}$, $\not{\underset{1}{u}} \to \not{\underset{2\,2}{u\dot{u}}}$, $\not{\underset{1}{\dot{u}}} \to \not{\underset{2\,2}{\ddot{u}\dot{u}}}$,
round 2	$\underset{2}{u} \to \underset{3\,3}{a\dot{a}}\,\underset{3\,3\,3}{u\dot{u}\ddot{u}}$, $\underset{2}{\dot{1}} \to \underset{3\,3}{1\dot{1}}$, $\underset{2}{\dot{0}} \to \underset{3\,3\,3}{0\dot{0}\ddot{0}}$, $\underset{2}{\ddot{u}} \to \epsilon$, $\underset{2}{a} \to \underset{3\,3}{a\dot{a}}\,\underset{3\,3}{a\dot{a}}$, $\underset{2}{\dot{a}} \to \epsilon$, $\not{\underset{2}{u}} \to \not{\underset{3\,3}{u\dot{u}}}$, $\not{\underset{2}{\dot{u}}} \to \not{\underset{3\,3}{u\dot{u}}}$, $\not{\underset{2}{\dot{u}}} \to \not{\underset{3\,3}{\ddot{u}\dot{u}}}$, $\not{\underset{2}{\ddot{u}}} \to \not{\underset{3\,3}{\ddot{u}\dot{u}}}$,
round 3	$\underset{3}{u} \to \underset{1\,1}{u\dot{u}}$, $\underset{3}{\dot{u}} \to \underset{1\,1}{u\dot{u}}$, $\underset{3}{\ddot{u}} \to \epsilon$, $\underset{3}{a} \to \underset{1\,1}{a\dot{a}}$, $\underset{3}{\dot{a}} \to \underset{1\,1}{a\dot{a}}$, $\not{\underset{3}{u}} \to \not{\underset{1\,1}{u\dot{u}}}$, $\not{\underset{3}{\dot{u}}} \to \not{\underset{1\,1}{u\dot{u}}}$, $\not{\underset{3}{\dot{u}}} \to \not{\underset{1\,1}{\ddot{u}\dot{u}}}$, $\not{\underset{3}{\ddot{u}}} \to \not{\underset{1\,1}{\ddot{u}\dot{u}}}$
Final round	$\underset{1}{\dot{u}} \to \underset{*\,*}{a\dot{a}}\,\underset{4\,4}{d\,\bar{u}\dot{\bar{u}}}$, $\underset{1}{\dot{a}} \to \underset{*\,*}{a\dot{a}}$, $\not{\underset{1}{\dot{u}}} \to \epsilon$, $\not{\underset{1}{\ddot{u}}} \to \underset{4\,4}{u\dot{u}}$, $d \to \epsilon$, $\underset{*}{\dot{a}} \to \underset{4\,4}{a\dot{a}}$

to $\not{\underset{j(2^k)+1}{\ddot{u}}}\not{\underset{j(2^k)+1}{\dot{u}}}$ and pairs of the form $\not{\underset{2^k+1}{u}}\not{\underset{2^k+1}{\dot{u}}}$ (parity pairs) will not be changed. So applying a rule to delete $\not{u}\not{\dot{u}}$ pairs after iteration $\lceil \log_2 n \rceil + c$ deletes all parity pairs from the dataword without deleting the $u_i\dot{u}_i = x_i\dot{x}_i$ pairs that appear in Equation (6).

Algorithm Details

From the lemma statement we begin with a word of the form

$$\underset{1}{u_1}z_1\ \underset{1}{u_2}z_2\ \underset{1}{u_3}z_3\ \underset{1}{u_4}z_4\ \cdots\ \underset{1}{u_{n+\lceil \log_2 n \rceil + c}}\,z_{n+\lceil \log_2 n \rceil + c} \tag{7}$$

Rules of the form $\underset{1}{u} \to \underset{2\,2\,2}{u\dot{u}\ddot{u}}$ are applied to Dataword (7) which after one round gives

$$\underset{2}{u_1}\underset{2}{\dot{u}_1}\underset{2}{\ddot{u}_1}\ \underset{2}{u_2}\underset{2}{\dot{u}_2}\underset{2}{\ddot{u}_2}\ \underset{2}{u_3}\underset{2}{\dot{u}_3}\underset{2}{\ddot{u}_3}\ \underset{2}{u_4}\underset{2}{\dot{u}_4}\underset{2}{\ddot{u}_4}\ \cdots\ \underset{2}{u_{n+\lceil \log_2 n \rceil + c}}\,\underset{2}{\dot{u}_{n+\lceil \log_2 n \rceil + c}}\,\underset{2}{\ddot{u}_{n+\lceil \log_2 n \rceil + c}} \tag{8}$$

The next round on the dataword uses the rules $\{\underset{2}{u} \to \underset{3\,3}{a\dot{a}}\,\underset{3\,3\,3}{u\dot{u}\ddot{u}}, \underset{2}{\dot{0}} \to \underset{3\,3\,3}{0\dot{0}\ddot{0}}, \underset{2}{\dot{1}} \to \underset{3\,3}{1\dot{1}}, \underset{2}{\ddot{u}} \to \epsilon\}$ to mark every second $\underset{2\,2\,2}{u\dot{u}\ddot{u}}$ triple in Dataword (8). Because 2-tag systems only read every second symbol, when $i = 1 \bmod 2$, symbols $\underset{2}{u_i}$ and $\underset{2}{\ddot{u}_i}$ are read applying the rules $\underset{2}{u} \to \underset{3\,3}{a\dot{a}}\,\underset{3\,3\,3}{u\dot{u}\ddot{u}}$ and $\underset{2}{\ddot{u}} \to \epsilon$ to append $\underset{3\,3}{a\dot{a}}\,\underset{3}{u_i}\underset{3}{\dot{u}_i}\underset{3}{\ddot{u}_i}$ (ϵ is the empty word), and when $i = 0 \bmod 2$, symbol $\underset{2}{\dot{u}_i}$ is read applying either the rule $\underset{2}{\dot{0}} \to \underset{3\,3\,3}{0\dot{0}\ddot{0}}$ or the rule $\underset{2}{\dot{1}} \to \underset{3\,3}{1\dot{1}}$ to append $\not{\underset{3}{u_i}}\not{\underset{3}{\dot{u}_i}}(\not{\underset{3}{\ddot{u}_i}})$ (the $\not{\underset{3}{\ddot{u}_i}}$ symbol in brackets is present only if $u_i = 0$). So the above rules mark every second $u\dot{u}\ddot{u}$ triple to give a dataword of one of the following two forms

$$\underset{3\,3}{a\dot{a}}\ \underset{3}{u_1}\underset{3}{\dot{u}_1}\underset{3}{\ddot{u}_1}\ \not{\underset{3}{u_2}}\not{\underset{3}{\dot{u}_2}}(\not{\underset{3}{\ddot{u}_2}})\ \underset{3\,3}{a\dot{a}}\ \underset{3}{u_3}\underset{3}{\dot{u}_3}\underset{3}{\ddot{u}_3}\ \not{\underset{3}{u_4}}\not{\underset{3}{\dot{u}_4}}(\not{\underset{3}{\ddot{u}_4}})\ \cdots\ \not{\underset{3}{u_{n+\lceil \log_2 n \rceil + c}}}\,\not{\underset{3}{\dot{u}_{n+\lceil \log_2 n \rceil + c}}}\,(\not{\underset{3}{\ddot{u}_{n+\lceil \log_2 n \rceil + c}}}) \tag{9}$$

$$\underset{3}{\dot{a}}\ \underset{3}{u_1}\underset{3}{\dot{u}_1}\underset{3}{\ddot{u}_1}\ \not{\underset{3}{u_2}}\not{\underset{3}{\dot{u}_2}}(\not{\underset{3}{\ddot{u}_2}})\ \underset{3\,3}{a\dot{a}}\ \underset{3}{u_3}\underset{3}{\dot{u}_3}\underset{3}{\ddot{u}_3}\ \not{\underset{3}{u_4}}\not{\underset{3}{\dot{u}_4}}(\not{\underset{3}{\ddot{u}_4}})\ \cdots\ \underset{3\,3}{a\dot{a}}\ \underset{3}{u_{n+\lceil \log_2 n \rceil + c}}\underset{3}{\dot{u}_{n+\lceil \log_2 n \rceil + c}}\underset{3}{\ddot{u}_{n+\lceil \log_2 n \rceil + c}} \tag{10}$$

We get a dataword of the form given in (9) if $n + \lceil \log_2 n \rceil + c$ is even and we get a dataword of the form given in (10) if $n + \lceil \log_2 n \rceil + c$ is odd. To see this recall from the previous paragraph that if $n + \lceil \log_2 n \rceil + c$ is odd we read $\underset{0}{\ddot{u}_{n+\lceil \log_2 n \rceil + c}}$, and when it is read it is deleted along with $\underset{3}{a_1}$ which is why $\underset{3}{a_1}$ does not appear at the left end of Dataword (10).

For the next round the rules $\{\underset{3}{u} \to \underset{1\,1}{u}\dot{\underset{1}{u}}, \dot{\underset{3}{u}} \to \underset{1\,1}{u}\dot{\underset{1}{u}}, \ddot{\underset{3}{u}} \to \epsilon, \not{\underset{3}{u}} \to \not{\underset{1\,1}{u}}\dot{\not{\underset{1}{u}}}, \dot{\not{\underset{3}{u}}} \to \underset{1\,1}{u}\dot{\underset{1}{u}}, \ddot{\not{\underset{3}{u}}} \to \epsilon, \underset{3}{a} \to \underset{1\,1}{a}\dot{\underset{1}{a}}, \dot{\underset{3}{a}} \to \underset{1\,1}{a}\dot{\underset{1}{a}},\}$ are applied to Datawords (9) and (10) to give Datawords (11) and (12), respectively. Note that to produce datawords of the form given in (11) and (12), where the leftmost symbol is $\underset{1}{a}$ instead of $\dot{\underset{1}{a}}$, Datawords (9) and (10) must have even length. The parity of Datawords (9) and (10) depends on the number of $1\dot{1}\ddot{1}$ triples that are marked when reading Dataword (8), as a $1\dot{1}\ddot{1}$ triple is marked by replacing with a $\not{1}\dot{\not{1}}$ pair. So if we mark an even number of $1\dot{1}\ddot{1}$ triples, Dataword (9) will have the same parity as Dataword (8), and Dataword (10) will have a different parity from Dataword (8) (since Dataword (10) is missing the leftmost $\underset{3}{a}_1$). So because Dataword (9) is only produced if Dataword (8) is even and Dataword (10) is only produced if Dataword (8) is odd, Dataword (9) and (10) are both even. From the description of Task (a) we know that the number of $\underset{2\,2\,2}{1\dot{1}\ddot{1}}$ triples marked is even for the first $\lceil \log_2 n \rceil + c$ iterations of our algorithm and so it follows that the length of Datawords (9) and (10) are even. For this reason the leftmost symbol in Datawords (11) and (12) is $\underset{1}{a}$ and so undotted symbols are read during the next round.

$$\underset{1\,1}{a}\dot{\underset{1}{a}}\ \underset{1\,1}{u_1}\dot{\underset{1}{u}}_1\ \not{\underset{1}{u}}_2\dot{\not{\underset{1}{u}}}_2\ \underset{1\,1}{a}\dot{\underset{1}{a}}\ \underset{1}{u_3}\dot{\underset{1}{u}}_3\ \not{\underset{1}{u}}_4\dot{\not{\underset{1}{u}}}_4\ \cdots\ \not{\underset{1}{u}}_{n+\lceil \log_2 n \rceil + c}\dot{\not{\underset{1}{u}}}_{n+\lceil \log_2 n \rceil + c} \qquad \text{if } n + \lceil \log_2 n \rceil + c \text{ is even} \tag{11}$$

$$\underset{1\,1}{a}\dot{\underset{1}{a}}\ \underset{1\,1}{u_1}\dot{\underset{1}{u}}_1\ \not{\underset{1}{u}}_2\dot{\not{\underset{1}{u}}}_2\ \underset{1\,1}{a}\dot{\underset{1}{a}}\ \underset{1}{u_3}\dot{\underset{1}{u}}_3\ \not{\underset{1}{u}}_4\dot{\not{\underset{1}{u}}}_4\ \cdots\ \underset{1\,1}{a}\dot{\underset{1}{a}}\ \underset{1}{u}_{n+\lceil \log_2 n \rceil + c}\dot{\underset{1}{u}}_{n+\lceil \log_2 n \rceil + c} \qquad \text{if } n + \lceil \log_2 n \rceil + c \text{ is odd} \tag{12}$$

In Datawords (11) and (12) the 2-tag system is ready to repeat the process of marking every second unmarked symbol. The cases for Datawords (11) and (12) proceed in a similar manner so we will continue only with the case for Dataword (11). Applying the rules $\underset{1}{u} \to \underset{2\,2\,2}{u}\dot{\underset{2}{u}}\ddot{\underset{2}{u}}$, $\not{\underset{1}{u}} \to \not{\underset{2\,2}{u}}\dot{\not{\underset{2}{u}}}$ and $\underset{1}{a} \to \underset{2\,2}{a}\dot{\underset{2}{a}}$ to Dataword (11) gives

$$\underset{2\,2}{a}\dot{\underset{2}{a}}\ \underset{2\,2}{u_1}\dot{\underset{2}{u}}_1\ddot{\underset{2}{u}}_1\ \not{\underset{2}{u}}_2\dot{\not{\underset{2}{u}}}_2\ \underset{2\,2}{a}\dot{\underset{2}{a}}\ \underset{2}{u_3}\dot{\underset{2}{u}}_3\ddot{\underset{2}{u}}_3\ \not{\underset{2}{u}}_4\dot{\not{\underset{2}{u}}}_4\ \underset{2\,2}{a}\dot{\underset{2}{a}}\ \underset{2}{u_5}\dot{\underset{2}{u}}_5\ddot{\underset{2}{u}}_5\ \cdots\ \not{\underset{2}{u}}_{n+\lceil \log_2 n \rceil + c}\dot{\not{\underset{2}{u}}}_{n+\lceil \log_2 n \rceil + c} \tag{13}$$

Continuing the computation, the rules used in the first iteration are used again here to mark every second $\underset{2\,2\,2}{u}\dot{\underset{2}{u}}\ddot{\underset{2}{u}}$ triple but on this iteration we also apply the rules $\{\not{\underset{2}{u}} \to \not{\underset{3\,3}{u}}\dot{\not{\underset{3}{u}}}, \dot{\not{\underset{2}{u}}} \to \not{\underset{3\,3}{u}}\dot{\not{\underset{3}{u}}}, \underset{2}{a} \to \underset{3\,3}{a}\dot{\underset{3}{a}}\ \underset{3\,3}{a}\dot{\underset{3}{a}}, \dot{\underset{2}{a}} \to \epsilon\}$ to Dataword (13) to produce Dataword (14). When reading Dataword (13) each triple $\underset{2\,2\,2}{u}\dot{\underset{2}{u}}\ddot{\underset{2}{u}}$ causes a shift in the reading frame where if we read the symbols with a single dot before a triple we will read the symbols with no dots after that triple. This means that if there is a even number of $\underset{2\,2\,2}{u}\dot{\underset{2}{u}}\ddot{\underset{2}{u}}$ triples to the left of a $\not{\underset{2\,2}{u}}\dot{\not{\underset{2}{u}}}$ pair we read $\not{\underset{2}{u}}$ and apply the rule $\not{\underset{2}{u}} \to \not{\underset{3\,3}{u}}\dot{\not{\underset{3}{u}}}$, and if there is an odd number we read $\dot{\not{\underset{2}{u}}}$ and apply the rule $\dot{\not{\underset{2}{u}}} \to \not{\underset{3\,3}{u}}\dot{\not{\underset{3}{u}}}$. This allows us to distinguish the parity pairs from all other pairs by changing as described in the paragraph on Task (c), since only the marked parity pairs have the form $\not{u}\dot{\not{u}}$ after the final iteration with all other marked pairs having the from $\not{u}\dot{\not{u}}$. Next we consider the change in the number of $a\dot{a}$ pairs when Dataword (13) is read to produce Dataword (14). If we read $\dot{\underset{2}{u}}$ in a $\underset{2}{u_i}\dot{\underset{2}{u}}_i\ddot{\underset{2}{u}}_i$ triple

it follows that we read $\underset{2}{\dot{a}}$ symbols in the run of $\underset{2}{a}\underset{2}{\dot{a}}$ pairs immediately to the left of that triple, and so when we mark a triple by applying the rule $\underset{2}{\dot{u}} \to \not{\underset{3}{u}}_2\dot{\not{\underset{3}{u}}}_2(\ddot{\not{\underset{3}{u}}}_2)$ the $\underset{2}{a}\underset{2}{\dot{a}}$ pairs are deleted because we apply the rule $\underset{2}{\dot{a}} \to \epsilon$ giving the behavior described in the third paragraph of Task (a). Alternatively, when we read $\underset{2}{u}$ the $\underset{3}{u}\underset{3}{\dot{u}}\underset{3}{\ddot{u}}$ triple remains unmarked and we read $\underset{2}{a}$ symbols in the run of $\underset{2}{a}\underset{2}{\dot{a}}$ pairs immediately to the left which applies the rule $\underset{2}{a} \to \underset{3}{a}\underset{3}{\dot{a}}\,\underset{3}{a}\underset{3}{\dot{a}}$ giving the described in the third paragraph of Task (a). Since we have covered the difference between reading even and odd numbers of unmarked symbols during the previous iteration, here we cover only the case where there is an even number of unmarked symbols in Dataword (13). So following a round on Dataword (13) we get a Dataword of the form given in (14).

$$(\underset{3}{a}\underset{3}{\dot{a}})^3\ \underset{3}{u}_1\underset{3}{\dot{u}}_1\underset{3}{\ddot{u}}_1\ \not{\underset{3}{u}}_2\dot{\not{\underset{3}{u}}}_2\ \not{\underset{3}{u}}_3\dot{\not{\underset{3}{u}}}_3(\ddot{\not{\underset{3}{u}}}_3)\ \not{\underset{3}{u}}_4\dot{\not{\underset{3}{u}}}_4\ (\underset{3}{a}\underset{3}{\dot{a}})^3\ \underset{3}{u}_5\underset{3}{\dot{u}}_5\underset{3}{\ddot{u}}_5\ \cdots\ \not{\underset{3}{u}}_{n+\lceil\log_2 n\rceil+c}\dot{\not{\underset{3}{u}}}_{n+\lceil\log_2 n\rceil+c} \tag{14}$$

From our explanation at the end of the previous iteration we know that a single round on Dataword (14) gives a dataword of the form

$$(\underset{1}{a}\underset{1}{\dot{a}})^3\ \underset{1}{u}_1\underset{1}{\dot{u}}_1\ \not{\underset{1}{u}}_2\dot{\not{\underset{1}{u}}}_2\ \not{\underset{1}{u}}_3\dot{\not{\underset{1}{u}}}_3\ \not{\underset{1}{u}}_4\dot{\not{\underset{1}{u}}}_4\ (\underset{1}{a}\underset{1}{\dot{a}})^3\ \underset{1}{u}_5\underset{1}{\dot{u}}_5\ \cdots\ \not{\underset{1}{u}}_{n+\lceil\log_2 n\rceil+c}\dot{\not{\underset{1}{u}}}_{n+\lceil\log_2 n\rceil+c} \tag{15}$$

Each subsequent iteration carries on as described above until we come to iteration $\lceil\log_2 n\rceil + c$. From Task (a) we know that during iteration $\lceil\log_2 n\rceil + c$ we mark an odd number of $\underset{2}{1}\dot{\underset{2}{1}}\ddot{\underset{2}{1}}$ triples and so from the paragraph preceding Dataword (9) this means that the leftmost $\underset{1}{a}$ gets deleted at the end of iteration $\lceil\log_2 n\rceil + c$ to give a Dataword of the form (16). From the description of Task (a) we know that after $\lceil\log_2 n\rceil + c$ iterations $\underset{1}{u}_1\underset{1}{\dot{u}}_1$ is the only unmarked pair in the dataword and that the only $a\dot{a}$ pairs to be found are immediately to the left of $\underset{1}{u}_1\underset{1}{\dot{u}}_1$ in a word of the form $(\underset{1}{a}\underset{1}{\dot{a}})^{2^{n+\lceil\log_2 n\rceil+c}-1}$ as shown in Dataword (16).

$$\underset{1}{\dot{a}}(\underset{1}{a}\underset{1}{\dot{a}})^{2^{n+\lceil\log_2 n\rceil+c}-2}\ \underset{1}{u}_1\underset{1}{\dot{u}}_1\ \not{\underset{1}{u}}_2\dot{\not{\underset{1}{u}}}_2\ \not{\underset{1}{u}}_3\dot{\not{\underset{1}{u}}}_3\ \not{\underset{1}{u}}_4\dot{\not{\underset{1}{u}}}_4\ \not{\underset{1}{u}}_5\dot{\not{\underset{1}{u}}}_5\ \cdots\ \not{\underset{1}{u}}_{n+\lceil\log_2 n\rceil+c}\dot{\not{\underset{1}{u}}}_{n+\lceil\log_2 n\rceil+c} \tag{16}$$

In Dataword (16) following $\lceil\log_2 n\rceil + c$ iterations we read the dotted symbol for pairs with underscript 1 for the first time and we apply the rules $\{\underset{1}{\dot{u}} \to \underset{*}{a}\underset{*}{\dot{a}}\,d\,\underset{4}{\bar{u}}\underset{4}{\dot{\bar{u}}},\ \underset{1}{\dot{a}} \to \underset{*}{a}\underset{*}{\dot{a}},\ \dot{\not{\underset{3}{u}}} \to \epsilon,\ \dot{\not{\underset{3}{u}}} \to \underset{4}{u}\underset{4}{\dot{u}},\ d \to \epsilon\}$ to give Dataword (17). From the description of Task (c) we know that rule $\dot{\not{\underset{3}{u}}} \to \epsilon$ deletes all pairs of the form $\not{\underset{4}{u}}_{2^k+1}\dot{\not{\underset{4}{u}}}_{2^k+1}$ and so from Equation (3) Datawords (17) and (6) are identical.

$$\underset{4}{\bar{u}}_1\underset{4}{\dot{\bar{u}}}_1\ \underset{4}{u}_4\underset{4}{\dot{u}}_4\ \underset{4}{u}_6\underset{4}{\dot{u}}_6\ \underset{4}{u}_7\underset{4}{\dot{u}}_7\ \underset{4}{u}_8\underset{4}{\dot{u}}_8\ \underset{4}{u}_{10}\underset{4}{\dot{u}}_{10}\ \cdots\ \underset{4}{u}_{n+\lceil\log_2 n\rceil+c}\underset{4}{\dot{u}}_{n+\lceil\log_2 n\rceil+c}(\underset{4}{a}\underset{4}{\dot{a}})^{2^{n+\lceil\log_2 n\rceil+c}} \tag{17}$$

◀

We can now use Lemma 11 to prove Corollary 12 which will be used in the reduction in our main theorem. In this reduction our tag systems simulate a particular type of Turing machine where the halting time is bounded by what is known as a longevity guard. A longevity guard [8] for a Turing machine M is a function $l : \Sigma^* \to \mathbb{N}$ where on input w either M halts in $\leqslant l(w)$ steps or it runs forever.

▶ **Corollary 12.** *Let M be an arbitrary nondeterministic Turing machine with a single binary tape and a longevity guard l, let f be the function given by Equation (3), and let s be an arbitrary binary word of length $|w| + \lceil\log_2 |w|\rceil + c$ where $c = 1$ if $2^{\lceil\log_2 |w|\rceil} < n + \lceil\log_2 |w|\rceil$ otherwise $c = 0$. There is a nondeterministic 2-tag system that takes words of the form $f(w, s)$, and halts in time $O(l(w)^2 \log l(w))$ if and only if M halts on input w in time $l(w)$.*

Proof. The deterministic tag system algorithm in [15] simulates a Turing machine transition rule q_y, x_1, x_2, d, q_z (with current state q_y, read symbol x_1, write symbol x_2, move direction $d \in \{L, R\}$, and next state q_z) using a rule of the form $\underset{y}{x_1}\dot{\underset{y}{x}}_1 \to \not{\underset{0}{x}}_2\dot{\not{\underset{0}{x}}}_2 s^{2^z}$ if $d = R$, and a rule of the form $\underset{y}{x_1}\dot{\underset{y}{x}}_1 \to s^{2^z}\underset{0}{x_2}\dot{\underset{0}{x}}_2$ if $d = L$. In our nondeterministic 2-tag system there is a rule of one of the two forms given above for each transition rule in the nondeterministic Turing machine M. Such a nondeterministic 2-tag system uses the same algorithm as the tag system in [15] and so simulates t steps of M in time $O(l(w)^2 \log l(w))$ (see Theorem 2) halting if and only if M halts. Adding the 2-tag system from Lemma 11 as a subroutine to such a nondeterministic 2-tag system allows it to simulate M in time $O(l(w)^2 \log l(w))$ when given $f(w, s)$ as input. ◀

To see that the longevity guard is necessary in Corollary 12 consider what happens if we replace $l(w)$ with an arbitrary running time t: Our 2-tag system still halts in time $O(t^2 \log t)$ if M halts in time t, however it is also possible that our tag system halts in time $O(t^2 \log t)$ when M does not halt in time t but does halt at some later time $> t$. This is because the asymptotic bound $O(t^2 \log t)$ does not bound precisely the number of simulated Turing machine steps at t. So without the longevity guard we get the "if" but not the "only if".

▶ **Theorem 13.** *The distributional bounded halting problem for nondeterministic 2-tag systems is complete for average-case NP under Ptime randomized reductions.*

Proof. We show that for every distributional problem (D, μ) in NP, there is a Ptime randomized reduction from (D, μ) to the distributional bounded halting problem for nondeterministic 2-tag systems that satisfies Definition 8.

In [8] Gurevich shows that for each NP distributional problem (D, μ) there exists a nondeterministic binary Turing machine M_i that has a polynomial longevity guard l such that (D, μ) Ptime reduces to the halting problem for M_i. Given a binary input word w that encodes an instance v of D, M_i halts in $\leqslant l(w)$ steps if and only if $v \in L_D$. From Definition 9 a Ptime function g that reduces instances of D to instances of the bounded halting problem for M_i is given by

$$g(v) = Pf(\langle i \rangle, w, 1^{l(w)}) \tag{18}$$

Let Δ be a dilation of (D, μ) to (D_Δ, μ_Δ) where each instance $v \in D$ is mapped to a set of the form $\{Pf(v, s_v) | s_v \in \{0,1\}^{|w|+\lceil \log_2 |w| \rceil + c}\}$, with $c = 1$ if $2^{\lceil \log_2 |w| \rceil} < |w| + \lceil \log_2 |w| \rceil$ otherwise $c = 0$. Since g is Ptime computable we know from Equation (18) that given v the value $|w| + \lceil \log_2 |w| \rceil + c$ can be computed in polynomial time and so there is a randomized algorithm that computes Δ giving a Ptime dilation. In addition Δ is also nonrare as $R_\Delta = 1$ (see Definition 7). So from Definition 8 we can show that (D, μ) Ptime randomly reduces to the bounded halting problem for 2-tag systems by showing that its nonrare Ptime dilation (D_Δ, μ_Δ) Ptime reduces to the bounded halting problem for 2-tag systems.

From Definition 6 and the paragraph before Equation (18) we see that for all $Pf(v, s_w) \in D_\Delta$ we have $Pf(v, s_w) \in L_{D_\Delta}$ if and only if M_i halts in time $l(w)$ on input w. It follows from Corollary 12 that there is a tag system T_j such that for all $Pf(v, s_w) \in D_\Delta$, T_j halts in time $O(l(w)^2 \log l(w))$ on input $f(v, s_w)$ if and only if $Pf(v, s_w) \in D_\Delta$. So there is a reduction g' that reduces instance of D_Δ to instances of the bounded halting problem for T_j (Definition 10). The reduction g' which is given in Equation (19) satisfies condition 1 of Definition 5.

$$g'(Pf(v, s_w)) = Pf(\langle j \rangle, f(w, s_w), 1^{O(l(w)^2 \log l(w))}) \tag{19}$$

Since $g(v)$ in Equation (18) is Ptime computable so too is $g'(Pf(v, s_w))$. Now to complete our proof it only remains to show that the reduction g' given in Equation (19) satisfies condition 2 of Definition 5.

We already know that the reduction g given in Equation (18) Ptime reduces the distributional problem (D, μ) to the bounded halting problem for M_i, and so it follows from condition 2 of Definition 5 and the probability distribution in Definition 9 that there is a polynomial p_1, such that

$$\mu(v) \ < \ p_1(|v|)2^{-(|\langle i\rangle|+|w|)}|\langle i\rangle|^{-2}|w|^{-2}l(w)^{-2} \tag{20}$$

For simplicity we rewrite Equation (21) as

$$\mu(v) \ < \ p_1(v)2^{-|w|} \tag{21}$$

From Equation (3) we get $|f(w, s_w)| = 2|w| + 2\log_2|w|$ and so from Definition 10 the probability of getting an instance $Pf(\langle j\rangle, f(w, s_w), 1^{O(l(w)^2 \log l(w))})$ is proportional to

$$\mu(\langle j\rangle, f(w, s_w), O(l(w)^2 \log l(w))) = \\ 2^{-(|\langle j\rangle|+2|w|+2\log_2|w|)}|\langle j\rangle|^{-2}(2|w| + 2\log_2|w|)^{-2}(l(w)^2 \log l(w))^{-2} \tag{22}$$

The value $|\langle j\rangle|$ is a constant independent of $|v|$ and the values $|w|$ and $l(w)$ are polynomial in $|v|$ and so there is a polynomial p_2 such that $p_2(|v|)^{-1} < 2^{(-(|\langle j\rangle|+2\log_2|w|)}|\langle j\rangle|^{-2}(2|w| + 2\log_2|w|)^{-2}(l(w)^2 \log l(w))^{-2}$ which substitutes into Equation (22) to give

$$\mu(\langle j\rangle, f(w, s_w), O(l(w)^2 \log l(w))) \ > \ p_2(v)^{-1}2^{-2|w|} \tag{23}$$

Recall that $|w_s| = |w|+\log_2|w|$ and so from Definition 6 an instance of D_Δ given by $Pf(v, w_s)$ has a probability proportional to

$$\mu_\Delta(Pf(v, w_s)) \ = \ \mu(v)2^{-(|w|+\log_2|w|)} \tag{24}$$

From Equations (21), (23) and (24) we get Equation (25) which shows that the polynomial $p_1(|v|)p_2(|v|)$ satisfies condition 2 of Definition 5 for the reduction g' in Equation (19).

$$\mu_\Delta(Pf(v, w_s)) \ < \ p_1(|v|)p_2(v)\mu(\langle j\rangle, f(w, s_w), O(l(w)^2 \log l(w))) \tag{25}$$

◀

4 An example of the decoding process, with paired notation

In this section we show a diagram of an example of the decoding process, using paired notation.

Each letter X in paired notation represents a pair of letters X_0X_1 in the tag system, of which only one will be read on the next round. Which one? That depends on the parity of the reading frame when it gets read. Often this parity is not known when the pair X_0X_1 is written.

Sometimes a third symbol λ is written to the tape, just for the purpose of changing the parity. If read, its rule appends an empty appendant. So it doesn't matter if it gets read or not.

In our paired notation, such a parity-changer is written as $\big|$ as in Figure 2.

The parity with which a pair is read is indicated by a thick line either over or under the symbol, as in Figure 2. So the vertical parts of the thick line are written by the previous line, while the horizontal parts simply alternate between the top and the bottom whenever there is a vertical part. They continue at the top (or bottom) from one line to the next.

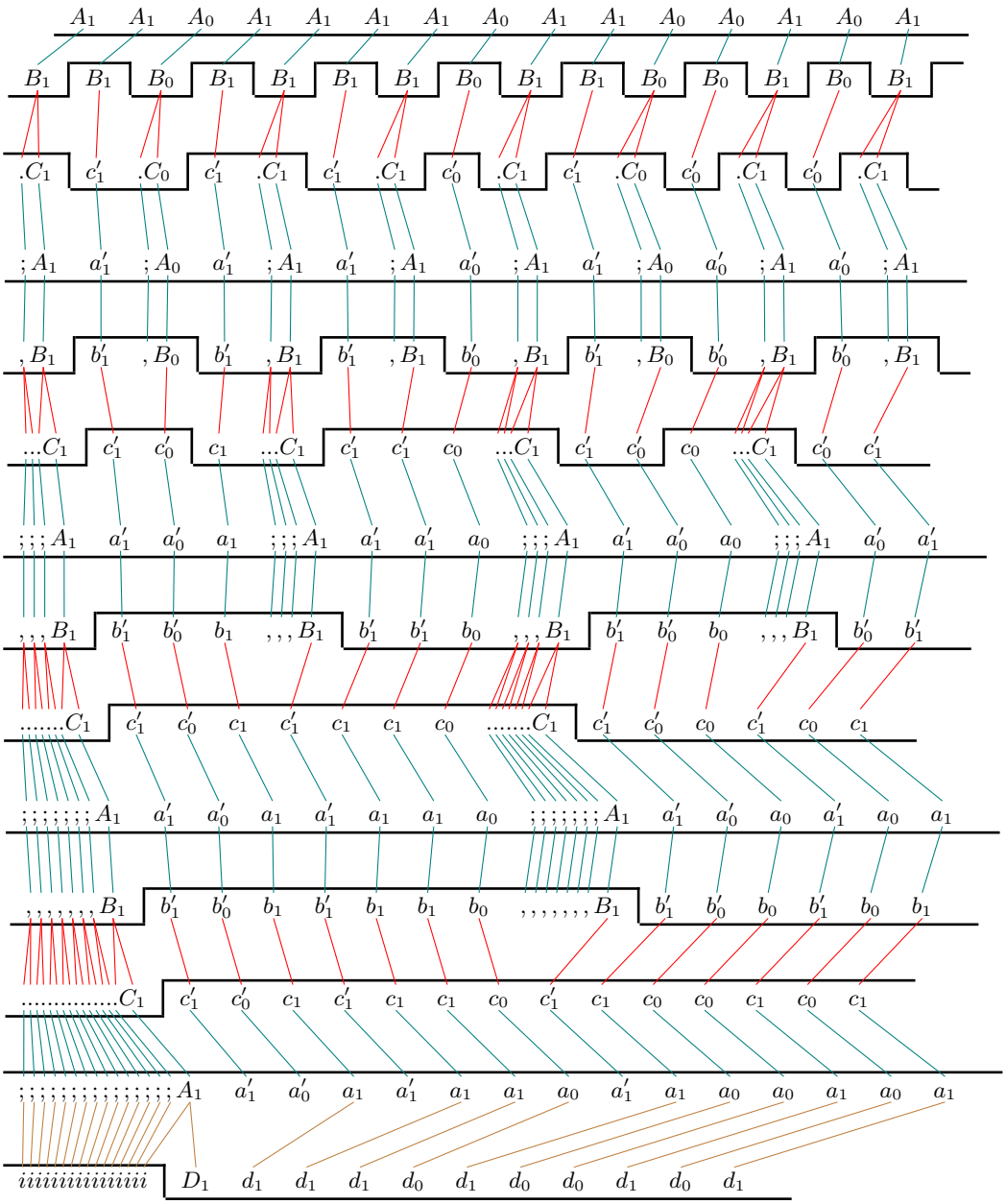

Figure 2 Diagram of an example of the decoding process. Each symbol here represents two symbols on the tag system tape, and the line passing over or under the symbols indicates the reading frame parity. The colored lines indicate how the symbols are transformed on each round. The bits marked with primes and eliminated in the final round are (if you trace them back to the initial row) exactly the extra bits used for the encoding. An additional partial round (not shown) over just the i symbols will yield the form guaranteed by Lemma 11: A row of bits, the first of which is uniquely marked, followed by a "counter" whose length is a power of two and which is at least as large as the row of bits, all being read in "plain" parity.

Table 2 The complete rules for the tag system. Paired notation is shown on the left; traditional notation is shown on the right. The initial tape would use either $\underset{1}{0}$ or $\underset{1}{1}$ as the second symbol in each A_i pair (not necessarily even matching the first symbol of the pair), but ever after the first round the second symbols of the A_i pairs will be as shown here. Since these input symbols will not be read, they make no difference. At the end, after a final partial round over just the $\underset{* *}{a\dot{a}}$ symbols, the correspondence of symbols in the final tape is $D_0 = \underset{4\,4}{\bar{0}\dot{\bar{0}}}$, $D_1 = \underset{4\,4}{\bar{1}\dot{\bar{1}}}$, $d_0 = \underset{4\,4}{0\dot{0}}$, $d_1 = \underset{4\,4}{1\dot{1}}$, and $e = \underset{4\,4}{a\dot{a}}$, and the tape is exactly the target dataword specified by Lemma 11.

symbol in diagram	*rule for plain parity*	*rule for flipped parity*	*corresponding original symbols and rules*		
A_0	$\underline{A_0} \longrightarrow B_0 \vert$	$\overline{A_0} \longrightarrow i \vert D_0$	$\underset{1\,1}{0\dot{0}}$	$\underset{1}{0} \to \underset{2\,2\,2}{0\dot{0}\ddot{0}}$	$\underset{1}{\dot{0}} \to \underset{* *}{a\dot{a}}\, d\, \underset{4\,4}{\bar{0}\dot{\bar{0}}}$
A_1	$\underline{A_1} \longrightarrow B_1 \vert$	$\overline{A_1} \longrightarrow i \vert D_1$	$\underset{1\,1}{1\dot{1}}$	$\underset{1}{1} \to \underset{2\,2\,2}{1\dot{1}\ddot{1}}$	$\underset{1}{\dot{1}} \to \underset{* *}{a\dot{a}}\, d\, \underset{4\,4}{\bar{1}\dot{\bar{1}}}$
a'_0	$\underline{a'_0} \longrightarrow b'_0$	$\overline{a'_0} \longrightarrow$	$\underset{1\,1}{\emptyset\dot{\emptyset}}$	$\underset{1}{\emptyset} \to \underset{2\,2}{\emptyset\dot{\emptyset}}$	$\underset{1}{\dot{\emptyset}} \to$
a'_1	$\underline{a'_1} \longrightarrow b'_1$	$\overline{a'_1} \longrightarrow$	$\underset{1\,1}{\not{1}\dot{\not{1}}}$	$\underset{1}{\not{1}} \to \underset{2\,2}{\not{1}\dot{\not{1}}}$	$\underset{1}{\dot{\not{1}}} \to$
a_0	$\underline{a_0} \longrightarrow b_0$	$\overline{a_0} \longrightarrow d_0$	$\underset{1\,1}{\not{\emptyset}\dot{\not{\emptyset}}}$	$\underset{1}{\not{\emptyset}} \to \underset{2\,2}{\not{\emptyset}\dot{\not{\emptyset}}}$	$\underset{1}{\dot{\not{\emptyset}}} \to \underset{4\,4}{0\dot{0}}$
a_1	$\underline{a_1} \longrightarrow b_1$	$\overline{a_1} \longrightarrow d_1$	$\underset{1\,1}{\not{1}\dot{\not{1}}}$	$\underset{1}{\not{1}} \to \underset{2\,2}{\not{1}\dot{\not{1}}}$	$\underset{1}{\dot{\not{1}}} \to \underset{4\,4}{1\dot{1}}$
;	$\underline{;} \longrightarrow ,$	$\bar{;} \longrightarrow i$	$\underset{1\,1}{a\dot{a}}$	$\underset{1}{a} \to \underset{* *}{a\dot{a}}$	$\underset{1}{\dot{a}} \to \underset{* *}{a\dot{a}}$
B_0	$\underline{B_0} \longrightarrow .C_0 \vert$	$\overline{B_0} \longrightarrow c'_0 \vert$	$\underset{2\,2}{0\dot{0}}$	$\underset{2}{0} \to \underset{3\,3\,3\,3\,3}{a\dot{a}0\dot{0}\ddot{0}}$	$\underset{2}{\dot{0}} \to \underset{3\,3\,3}{\emptyset\dot{\emptyset}\ddot{\emptyset}}$
B_1	$\underline{B_1} \longrightarrow .C_1 \vert$	$\overline{B_1} \longrightarrow c'_1$	$\underset{2\,2}{1\dot{1}}$	$\underset{2}{1} \to \underset{3\,3\,3\,3\,3}{a\dot{a}1\dot{1}\ddot{1}}$	$\underset{2}{\dot{1}} \to \underset{3\,3}{\not{1}\dot{\not{1}}}$
b'_0	$\underline{b'_0} \longrightarrow c_0$	$\overline{b'_0} \longrightarrow c'_0$	$\underset{2\,2}{\emptyset\dot{\emptyset}}$	$\underset{2}{\emptyset} \to \underset{3\,3}{\not{\emptyset}\dot{\not{\emptyset}}}$	$\underset{2}{\dot{\emptyset}} \to \underset{3\,3}{\emptyset\dot{\emptyset}}$
b'_1	$\underline{b'_1} \longrightarrow c_1$	$\overline{b'_1} \longrightarrow c'_1$	$\underset{2\,2}{\not{1}\dot{\not{1}}}$	$\underset{2}{\not{1}} \to \underset{3\,3}{\not{1}\dot{\not{1}}}$	$\underset{2}{\dot{\not{1}}} \to \underset{3\,3}{\not{1}\dot{\not{1}}}$
b_0	$\underline{b_0} \longrightarrow c_0$	$\overline{b_0} \longrightarrow c_0$	$\underset{2\,2}{\not{\emptyset}\dot{\not{\emptyset}}}$	$\underset{2}{\not{\emptyset}} \to \underset{3\,3}{\not{\emptyset}\dot{\not{\emptyset}}}$	$\underset{2}{\dot{\not{\emptyset}}} \to \underset{3\,3}{\not{\emptyset}\dot{\not{\emptyset}}}$
b_1	$\underline{b_1} \longrightarrow c_1$	$\overline{b_1} \longrightarrow c_1$	$\underset{2\,2}{\not{1}\dot{\not{1}}}$	$\underset{2}{\not{1}} \to \underset{3\,3}{\not{1}\dot{\not{1}}}$	$\underset{2}{\dot{\not{1}}} \to \underset{3\,3}{\not{1}\dot{\not{1}}}$
,	$\underline{:} \longrightarrow ..$	$\bar{:} \longrightarrow$	$\underset{2\,2}{a\dot{a}}$	$\underset{2}{a} \to \underset{3\,3\,3\,3}{a\dot{a}a\dot{a}}$	$\underset{2}{\dot{a}} \to$
C_0	$\underline{C_0} \longrightarrow A_0$	$\overline{C_0} \longrightarrow A_0$	$\underset{3\,3}{0\dot{0}}$	$\underset{3}{0} \to \underset{1\,1}{0\dot{0}}$	$\underset{3}{\dot{0}} \to \underset{1\,1}{0\dot{0}}$
C_1	$\underline{C_1} \longrightarrow A_1$	$\overline{C_1} \longrightarrow A_1$	$\underset{3\,3}{1\dot{1}}$	$\underset{3}{1} \to \underset{1\,1}{1\dot{1}}$	$\underset{3}{\dot{1}} \to \underset{1\,1}{1\dot{1}}$
c'_0	$\underline{c'_0} \longrightarrow a'_0$	$\overline{c'_0} \longrightarrow a'_0$	$\underset{3\,3}{\emptyset\dot{\emptyset}}$	$\underset{3}{\emptyset} \to \underset{1\,1}{\emptyset\dot{\emptyset}}$	$\underset{3}{\dot{\emptyset}} \to \underset{1\,1}{\emptyset\dot{\emptyset}}$
c'_1	$\underline{c'_1} \longrightarrow a'_1$	$\overline{c'_1} \longrightarrow a'_1$	$\underset{3\,3}{\not{1}\dot{\not{1}}}$	$\underset{3}{\not{1}} \to \underset{1\,1}{\not{1}\dot{\not{1}}}$	$\underset{3}{\dot{\not{1}}} \to \underset{1\,1}{\not{1}\dot{\not{1}}}$
c_0	$\underline{c_0} \longrightarrow a_0$	$\overline{c_0} \longrightarrow a_0$	$\underset{3\,3}{\not{\emptyset}\dot{\not{\emptyset}}}$	$\underset{3}{\not{\emptyset}} \to \underset{1\,1}{\not{\emptyset}\dot{\not{\emptyset}}}$	$\underset{3}{\dot{\not{\emptyset}}} \to \underset{1\,1}{\not{\emptyset}\dot{\not{\emptyset}}}$
c_1	$\underline{c_1} \longrightarrow a_1$	$\overline{c_1} \longrightarrow a_1$	$\underset{3\,3}{\not{1}\dot{\not{1}}}$	$\underset{3}{\not{1}} \to \underset{1\,1}{\not{1}\dot{\not{1}}}$	$\underset{3}{\dot{\not{1}}} \to \underset{1\,1}{\not{1}\dot{\not{1}}}$
.	$\underline{\cdot} \longrightarrow ;$	$\bar{\cdot} \longrightarrow ;$	$\underset{3\,3}{a\dot{a}}$	$\underset{3}{a} \to \underset{1\,1}{a\dot{a}}$	$\underset{3}{\dot{a}} \to \underset{1\,1}{a\dot{a}}$
i	$\underline{i} \longrightarrow e$	$\bar{i} \longrightarrow e$	$\underset{* *}{a\dot{a}}$	$\underset{*}{a} \to \underset{4\,4}{a\dot{a}}$	$\underset{*}{\dot{a}} \to \underset{4\,4}{a\dot{a}}$

The production rules in this notation therefore need to include the thick line either over or under the symbol on the left hand side, but on the right hand side only vertical thick lines can appear, no horizontal ones.

The correspondence between the symbols used in Figure 2 (which are compressed due to space constraints in the figure) and the symbols used in the paper is given in Table 2.

References

1 Andreas Blass and Yuri Gurevich. Matrix Transformation Is Complete for the Average Case. *SIAM Journal on Computing*, 24(1):3–29, 1995. doi:10.1137/S0097539792232070.

2 Vincent D Blondel and John N Tsitsiklis. A survey of computational complexity results in systems and control. *Automatica*, 36:1249–1274, 2000. doi:10.1016/S0005-1098(00)00050-9.

3 Andrej Bogdanov and Luca Trevisan. Average-Case Complexity. *CoRR*, abs/cs/0606037, 2006. arXiv:cs/0606037.

4 Matthew Cook. Universality in elementary cellular automata. *Complex Systems*, 15(1):1–40, 2004. URL: http://www.complex-systems.com/abstracts/v15_i01_a01.html.

5 Matthew Cook and Turlough Neary. A New Proof of Average Case NP-Completeness for the Post Correspondence Problem. In preparation.

6 Matthew Cook and Turlough Neary. Universality and Average-Case NP-Completeness in 2D Collatz Functions and Piecewise Affine Functions. In preparation.

7 Jens Eisert, Markus P Müller, and Christian Gogolin. Quantum measurement occurrence is undecidable. *Physical Review Letters*, 108(26):260501, 2012. doi:10.1103/PhysRevLett.108.260501.

8 Yuri Gurevich. Average case completeness. *Journal of Computer and System Sciences*, 42:346–398, 1991. doi:10.1016/0022-0000(91)90007-R.

9 Tero Harju and Maurice Margenstern. Splicing systems for universal Turing machines. In *DNA Computing, 10th International Workshop on DNA Computing(2004)*, volume 3384 of *Lecture Notes in Computer Science*, pages 149–158. Springer, 2005. doi:10.1007/11493785_13.

10 Pascal Koiran and Cristopher Moore. Closed-form analytic maps in one and two Dimensions can simulate universal Turing machines. *Theoretical Computer Science*, 210(1):217–223, 1999. doi:10.1016/S0304-3975(98)00117-0.

11 Leonid Levin. Average case complete problems. *SIAM Journal on Computing*, 15(1):285–1286, 1986. doi:10.1137/0215020.

12 Kristian Lindgren and Mats G. Nordahl. Universal Computation in Simple One-Dimensional Cellular Automata. *Complex Systems*, 4(3):299–318, 1990. URL: http://www.complex-systems.com/abstracts/v04_i03_a04.html.

13 Yuri Matiyasevich and Géraud Sénizergues. Decision problems for semi-Thue systems with a few rules. *Theoretical Computer Science*, 330(2):145–169, 2005. doi:10.1016/j.tcs.2004.09.016.

14 Marvin Minsky. Size and structure of universal Turing machines using tag systems. In *Recursive Function Theory, Symposium in Pure Mathematics*, volume 5, pages 229–238, Provelence, 1962. AMS.

15 Turlough Neary. *Small universal Turing machines*. PhD thesis, Department of Computer Science, National University of Ireland, Maynooth, 2008.

16 Turlough Neary. Undecidability in Binary Tag Systems and the Post Correspondence Problem for Five Pairs of Words. In *32nd International Symposium on Theoretical Aspects of Computer Science, STACS*, volume 30 of *LIPIcs*, pages 649–661, 2015. doi:10.4230/LIPIcs.STACS.2015.649.

17 Turlough Neary and Damien Woods. ¶-completeness of cellular automaton Rule 110. In *International Colloquium on Automata, Languages and Programming 2006, (ICALP) Part I*, volume 4051 of *Lecture Notes in Computer Science*, pages 132–143. Springer, 2006. doi:10.1007/11786986_13.

18 Emil Post. Formal reductions of the general combinatorial decision problem. *American Journal of Mathematics*, 65(2):197–215, 1943. doi:10.2307/2371809.

19 Raphael Robinson. Undecidability and nonperiodicity for tilings of the plane. *Inventiones Mathematicae*, 12(3):177–209, 1971. doi:10.1007/BF01418780.

20 Yurii Rogozhin. Small universal Turing machines. *Theoretical Computer Science*, 168(2):215–240, 1996. doi:10.1016/S0304-3975(96)00077-1.

21 Yurii Rogozhin and Sergey Verlan. On the rule complexity of universal tissue P systems. In *Sixth international Workshop on Membrane Computing(2005)*, volume 3850 of *Lecture Notes in Computer Science*, pages 356–362. Springer, 2006. doi:10.1007/11603047_24.

22 Paul Rothemund. A DNA and restriction enzyme implementation of Turing Machines. In *DNA Based Computers: Proceeding of a DIMACS Workshop*, volume 2055, pages 75–119. AMS, 1996. URL: https://authors.library.caltech.edu/27384/.

23 Hava Siegelmann and Maurice Margenstern. Nine switch-affine neurons suffice for Turing universality. *Neural Networks*, 12:593–600, 1999. doi:10.1016/S0893-6080(99)00025-8.

24 Hava Siegelmann and Eduardo Sontag. On the computational power of neural nets. *Journal of Computer and System Sciences*, 50(1):132–150, 1995. doi:10.1006/jcss.1995.1013.

25 Jie Wang. Average-case computational complexity theory. In Lane A Hemaspaandra and Alan L Selman, editors, *Complexity theory retrospective II*, pages 295–328. Springer-Verlag, 1998.

26 Damien Woods and Turlough Neary. On the time complexity of 2-tag systems and small universal Turing machines. In *In 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 132–143, Berkeley, California, October 2006. IEEE. doi:10.1109/FOCS.2006.58.

27 Adam Yedidia and Scott Aaronson. A Relatively Small Turing Machine Whose Behavior Is Independent of Set Theory. *Complex Systems*, 25(4):297–237, 2016. URL: http://www.complex-systems.com/abstracts/v25_i04_a04.html.